Bộ môn An toàn Thông tin – Khoa Mạng máy tính và Truyền thông
Trường Đại học Công nghệ Thông tin – Đại học Quốc gia thành phố Hồ Chí Minh

3

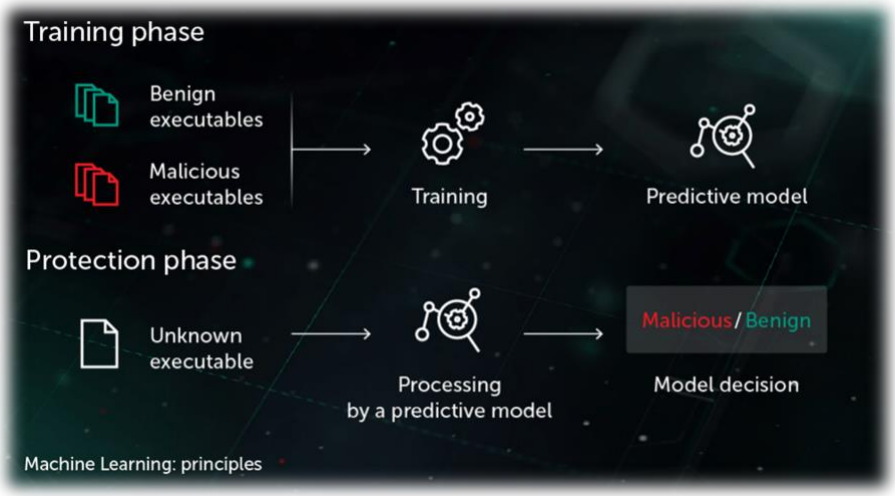
Lab

PHỤC VỤ MỤC ĐÍCH GIÁO DỤC
FOR EDUCATIONAL PURPOSE ONLY

Advanced Malware Detection

Python

Thực hành môn Phương pháp học máy trong an toàn thông tin



Tháng 10/2020
Lưu hành nội bộ

Mọi góp ý về tài liệu, vui lòng gửi về email inseclab@uit.edu.vn

A. TỔNG QUAN

1. Mục tiêu

- Tiếp cận các phần mềm độc hại được làm rối mã và đóng gói, mở rộng tính năng cho N-gram.
- Sử dụng học sâu để phát hiện và tạo phần mềm độc hại.

2. Thời gian thực hành

- Thực hành tại lớp: **5** tiết tại phòng thực hành.
- Hoàn thành báo cáo kết quả thực hành: tối đa **14** ngày.

3. Môi trường thực hành

> Google Colab (https://research.google.com/colaboratory/) – *khuyến khích không bắt buộc* và thư mục tải tài nguyên

- Keras
- TensorFlow
- XGBoost
- UPX
- Statsmodels

B. THỰC HÀNH

1. Phát hiện Javascript bị rối mã

Giải nén tập tin `JavascriptSamplesNotObfuscated.7z` và `JavascriptSamplesObfuscated.7z` thành thư 2 thư mục `JavaScript Samples` và `JavaScript Samples Obfuscated`.

1. Import các thư viện cần thiết để xử lý nội dung JavaScript, chuẩn bị tập dữ liệu, phân loại và đo hiệu suất bộ phân loại.

```
import os
from sklearn.feature_extraction.text import HashingVectorizer,
TfidfTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.pipeline import Pipeline
```

2. Định nghĩa đường dẫn 2 thư mục Javascipts đã giải nén.

```
js_path = "JavascriptSamples"
obfuscated_js_path = "ObfuscatedJavascriptSamples"

corpus = []
labels = []

file_types_and_labels = [(js_path, 0), (obfuscated_js_path, 1)]
```

3. Tiếp theo ta sẽ gán nhãn cho chúng.

```
for files_path, label in file_types_and_labels:
    files = os.listdir(files_path)
    for file in files:
        file_path = files_path + "/" + file
        try:
            with open(file_path, "r") as myfile:
                data = myfile.read().replace("\n", "")
                data = str(data)
                corpus.append(data)
                labels.append(label)
        except:
            pass
```

4. Ta chia tập dữ liệu thành tập huấn luyện và tập thử nghiệm, đồng thời tạo pipeline cho NLP, tiếp theo sử dụng phân loại random forest.

```
X_train, X_test, y_train, y_test = train_test_split(
    corpus, labels, test_size=0.33, random_state=42
)
text_clf = Pipeline(
    [
        ("vect",   HashingVectorizer(input="content",   ngram_range=(1,
3))),
        ("tfidf", TfidfTransformer(use_idf=True,)),
        ("rf", RandomForestClassifier(class_weight="balanced")),
    ]
)
```

5. Sau đó chạy huấn luyện và cho ra đánh giá.

```
text_clf.fit(X_train, y_train)
```

```
y_test_pred = text_clf.predict(X_test)

print(accuracy_score(y_test, y_test_pred))
print(confusion_matrix(y_test, y_test_pred))
```

® **Bài tập (yêu cầu làm)**

1. *Cho biết kết quả accuracy và confusion matrix.*

2. Trích xuất thuộc tính tập tin PDF

Sử dụng PDFiD Python (Didier Stevens - https://blog.didierstevens.com). Stevens đã chọn một danh sách gồm 20 thuộc tính thường được tìm thấy trong tập tin độc hại, gồm các tập tin PDF chứa các JavaScript được chạy tự động. Công cụ sẽ đếm số lần xuất hiện của thuộc tính

```
┌──(kali㉿kali)-[~/Desktop]
└─$ pdfid PythonBrochure.pdf
PDFiD 0.2.8 PythonBrochure.pdf
 PDF Header: %PDF-1.6
 obj                    1096
 endobj                 1095
 stream                 1061
 endstream              1061
 xref                      0
 trailer                   0
 startxref                 2
 /Page                    32
 /Encrypt                  0
 /ObjStm                  43
 /JS                       0
 /JavaScript               0
 /AA                       1
 /OpenAction               0
 /AcroForm                 1
 /JBIG2Decode              0
 /RichMedia                0
 /Launch                   0
 /EmbeddedFile             0
 /XFA                      0
 /Colors > 2^24            0
```

1. Import IPython để thu thập các output của script.

```
from IPython.utils import io
```

2. Định nghĩa hàm trích xuất thuộc tính. Chạy pdfid đọc một tập và lấy kết quả output của chúng. Kế tiếp, phân tích output để lấy vector số.

```
def PDF_to_FV(file_path):
    """Featurize a PDF file using pdfid."""
    with io.capture_output() as captured:
        %run -i pdfid $file_path
    out = captured.stdout
    out1 = out.split("\n")[2:-2]
    return [int(x.split()[-1]) for x in out1]
```

3. Import `listdir` để liệt kê các tập tin của thư mục PDF

```
from os import listdir
PDFs_path = "PDFSamples"
```

4. Cho vào vòng lặp để trích xuất, quét hết tất cả tập tin vào mảng `X`.

```
X = []
files = listdir(PDFs_path)
for file in files:
    file_path = PDFs_path + file
    X.append(PDF_to_FV(file_path))
print(X)
```

® **Bài tập (yêu cầu làm)**

2. Cho biết kết quả vector X.

3. Trích xuất N-grams bằng cách sử dụng thuật toá hash-gram

1. Chỉ định thư mục cần trích xuất, tham số N, import thư viện để hash và trích xuất N-grams từ chuỗi.

```
from os import listdir
from nltk import ngrams
import hashlib

directories = ["Benign PE Samples", "Malicious PE Samples"]
```

```
N = 2
```

2. Tạo các hàm đọc tập tin và chuyển chúng thành N-grams.

```
def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data

def byte_sequence_to_Ngrams(byte_sequence, N):
    """Creates a list of N-grams from a byte sequence."""
    return ngrams(byte_sequence, N)
```

3. Tiến hành hash N-grams.

```
def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data

def byte_sequence_to_Ngrams(byte_sequence, N):
    """Creates a list of N-grams from a byte sequence."""
    return ngrams(byte_sequence, N)
```

4. Hàm hash_file_Ngrams_into_dictionary lấy một N-grams, hash nó, sau đó tăng số lượng count trong dict cho hàm băm. Module B đảm bảo không thể có nhiều hơn B khoá trong dict.

```
def hash_file_Ngrams_into_dictionary(file_Ngrams, T):
    """Hashes N-grams in a list and then keeps track of the counts in a
dictionary."""
    for Ngram in file_Ngrams:
        hashable_Ngram = make_ngram_hashable(Ngram)
        hashed_and_reduced = hash_input(hashable_Ngram) % B
        T[hashed_and_reduced] = T.get(hashed_and_reduced, 0) + 1
```

5. Giá trị B là số nguyên tố lớn nhất nhỏ hơn 2^16 và tạo dict rỗng. Tiếp theo lặp lại qua các tập tin để count N-grams đã hash.

```
B = 65521
T = {}
for dataset_path in directories:
    samples = [f for f in listdir(dataset_path)]
    for file in samples:
        file_path = dataset_path + "/" + file
        file_byte_sequence = read_file(file_path)
        file_Ngrams = byte_sequence_to_Ngrams(file_byte_sequence, N)
        hash_file_Ngrams_into_dictionary(file_Ngrams, T)
```

6. Tạ chọn 1000 N-gram phổ biến sử dụng với heapq.

```
K1 = 1000
import heapq

K1_most_common_Ngrams_Using_Hash_Grams = heapq.nlargest(K1, T)
```

7. Sau khi chọn top N-grams được băm, ta tạo bộ thuộc tính N-grams, làm tăng vector đặc trưng.

```
def featurize_sample(file, K1_most_common_Ngrams_Using_Hash_Grams):
    """Takes a sample and produces a feature vector.
    The features are the counts of the K1 N-grams we've selected.
    """
    K1 = len(K1_most_common_Ngrams_Using_Hash_Grams)
    fv = K1 * [0]
    file_byte_sequence = read_file(file_path)
    file_Ngrams = byte_sequence_to_Ngrams(file_byte_sequence, N)
    for Ngram in file_Ngrams:
        hashable_Ngram = make_ngram_hashable(Ngram)
        hashed_and_reduced = hash_input(hashable_Ngram) % B
        if                    hashed_and_reduced                    in
K1_most_common_Ngrams_Using_Hash_Grams:
            index                                                    =
K1_most_common_Ngrams_Using_Hash_Grams.index(hashed_and_reduced)
            fv[index] += 1
    return fv
```

8. Cuối cùng tạo bộ dữ liệu.

```
X = [] for dataset_path in directories: samples = [f for f in
listdir(dataset_path)] for file in samples: file_path = dataset_path +
```

```
"/" + file X.append(featurize_sample(file_path,
K1_most_common_Ngrams_Using_Hash_Grams))
```

® **Bài tập (yêu cầu làm)**

3. Cho biết kết quả vector X.

4. Xây dựng bộ phân loại động phần mềm độc hại

Sử dụng bộ tập dữ liệu VirusShare cho các ứng dụng Android. Phân tích này đã được thực hiện trên LG Nexus 5 với Android API 23 (4000 ứng dụng độc hại và 4300 ứng dụng lành tính).

1. Do tập dữ liệu là các tập tin nhật ký định JSON nên import thư viện json

```
import numpy as np
import os
import json

directories_with_labels = [("DA Logs Benign", 0), ("DA Logs Malware",
1)]
```

2. Viết hàm parse nhật ký JSON.

```
def get_API_class_method_type_from_log(log):
    """Parses out API calls from behavioral logs."""
    API_data_sequence = []
    with open(log) as log_file:
        json_log = json.load(log_file)
        api_calls_array = "[" + json_log["api_calls"] + "]"
```

3. Chọn trích xuất class, method và type từ API call.

```
        api_calls = json.loads(api_calls_array)
        for api_call in api_calls:
            data = api_call["class"] + ":" + api_call["method"] + ":" +
api_call["type"]
            API_data_sequence.append(data)
    return API_data_sequence
```

4. Sau đó phân nhãn.

```
data_corpus = []
labels = []
for directory, label in directories_with_labels:
    logs = os.listdir(directory)
    for log_path in logs:
        file_path = directory + "/" + log_path
        try:

data_corpus.append(get_API_class_method_type_from_log(file_path))
            labels.append(label)
        except:
            pass
print(data_corpus[0])
```

5. Chia tập dữ liệu huấn luyện và kiểm thử

```
from sklearn.model_selection import train_test_split

corpus_train, corpus_test, y_train, y_test = train_test_split(
    data_corpus, labels, test_size=0.2, random_state=11
)
```

6. Trích xuất N-grams.

```
import collections
from nltk import ngrams
import numpy as np

def read_file(file_path):
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data

def text_to_Ngrams(text, n):
    """Produces a list of N-grams from a text."""
    Ngrams = ngrams(text, n)
    return list(Ngrams)
```

```
def get_Ngram_counts(text, N):
    """Get a frequency count of N-grams in a text."""
    Ngrams = text_to_Ngrams(text, N)
    return collections.Counter(Ngrams)
```

7. Xác định N=4 và thu thập tất cả N-grams.

```
N = 4
total_Ngram_count = collections.Counter([])
for file in corpus_train:
    total_Ngram_count += get_Ngram_counts(file, N)
```

8. Sau đó thu hẹp K1=3000 các N-grams phổ biến.

```
K1 = 3000
K1_most_frequent_Ngrams = total_Ngram_count.most_common(K1)
K1_most_frequent_Ngrams_list = [x[0] for x in K1_most_frequent_Ngrams]
```

9. Viết một hàm phan lại một mẫu thành một vector có N-grams.

```
def featurize_sample(file, Ngrams_list):
    """Takes a sample and produces a feature vector.
    The features are the counts of the K1 N-grams we've selected.
    """
    K1 = len(Ngrams_list)
    feature_vector = K1 * [0]
    fileNgrams = get_Ngram_counts(file, N)
    for i in range(K1):
        feature_vector[i] = fileNgrams[Ngrams_list[i]]
    return feature_vector
```

10. Sử dụng hàm trên để tạo bộ huấn luyện và mẫu kiểm thử.

```
X_train = []
for sample in corpus_train:
    X_train.append(featurize_sample(sample,
K1_most_frequent_Ngrams_list))
X_train = np.asarray(X_train)
X_test = []
for sample in corpus_test:
```

```
    X_test.append(featurize_sample(sample,
K1_most_frequent_Ngrams_list))
X_test = np.asarray(X_test)
```

11. Tiếp tục thu hẹp K1=3000 N-12 grams thành K2=500. Sau đó thiết lập pipeline chạy phân loại XGBoost.

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier

K2 = 500
mi_pipeline = Pipeline(
    [
        ("mutual_information", SelectKBest(mutual_info_classif, k=K2)),
        ("xgb", XGBClassifier()),
    ]
)
```

12. Sau đó huấn luyện và kiểm thử tập train, test.

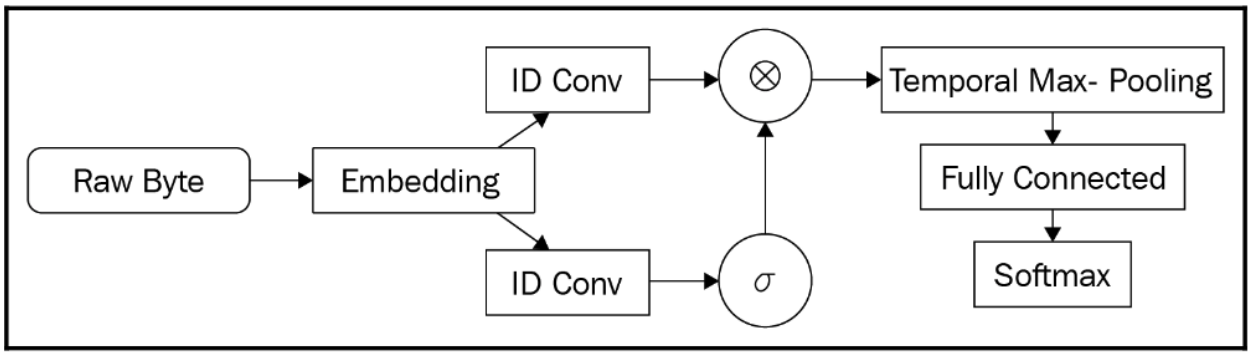
```
mi_pipeline.fit(X_train, y_train)
print("Training accuracy:")
print(mi_pipeline.score(X_train, y_train))
print("Testing accuracy:")
print(mi_pipeline.score(X_test, y_test))
```

® **Bài tập (yêu cầu làm)**

4. Cho biết kết quả đánh giá.

5. MalConv – Quy trình áp dụng sâu cho phát hiện phần mềm độc hại PE

Phương pháp này sử dụng raw byte đưa vào mạng thần kinh tích chập để huấn luyện (https://arxiv.org/pdf/1710.09435.pdf). Hệ thống kiến trúc của MalConv như hình sau:



1. Import thư viên numpy để tính toán vector và tqdm để theo dõi tiền trình trong vòng lặp.

```
import numpy as np
from tqdm import tqdm
```

2. Định nghĩa hàm để chuyển byte thành vector.

```
def embed_bytes(byte):
    binary_string = "{0:08b}".format(byte)
    vec = np.zeros(8)
    for i in range(8):
        if binary_string[i] == "1":
            vec[i] = float(1) / 16
        else:
            vec[i] = -float(1) / 16
    return vec
```

3. Đọc các tin PE mẫu và dán nhãn cho chúng.

```
import os
from os import listdir

directories_with_labels = [("Benign PE Samples", 0), ("Malicious PE
Samples", 1)]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for file in samples:
        file_path = os.path.join(dataset_path, file)
        list_of_samples.append(file_path)
```

```
        labels.append(label)
```

4. Định nghĩa hàm đọc chuỗi byte trong tập tin.

```
def read_file(file_path):
    """Read the binary sequence of a file."""
    with open(file_path, "rb") as binary_file:
        return binary_file.read()
```

5. Đặt độ dài tối đa, maxSize byte, để đọc cho mỗi mẫu, lấy tất cả byte của mẫu đưa vào X.

```
max_size = 15000
num_samples = len(list_of_samples)
X = np.zeros((num_samples, 8, max_size))
Y = np.asarray(labels)
file_num = 0
for file in tqdm(list_of_samples):
    sample_byte_sequence = read_file(file)
    for i in range(min(max_size, len(sample_byte_sequence))):
        X[file_num, :, i] = embed_bytes(sample_byte_sequence[i])
    file_num += 1
```

6. Thiết lập trình tối ưu.

```
from keras import optimizers
my_opt = optimizers.SGD(lr=0.01, decay=1e-5, nesterov=True)
```

7. Sử dụng API của Keras để thiết lập màn thần kinh học sâu.

```
from keras import Input
inputs = Input(shape=(8, max_size))

from keras.layers import Conv1D
conv1    =    Conv1D(kernel_size=(128),    filters=32,    strides=(128),
padding="same")(inputs)
conv2    =    Conv1D(kernel_size=(128),    filters=32,    strides=(128),
padding="same")(inputs)

from keras.layers import Activation
a = Activation("sigmoid", name="sigmoid")(conv2)
```

```
from keras.layers import multiply
mul = multiply([conv1, a])
b = Activation("relu", name="relu")(mul)

from keras.layers import GlobalMaxPool1D
p = GlobalMaxPool1D()(b)

from keras.layers import Dense
d = Dense(16)(p)
predictions = Dense(1, activation="sigmoid")(d)

from keras import Model
model = Model(inputs=inputs, outputs=predictions)
```

8. Biên dịch mô hình và chọn batch size.

```
model.compile(optimizer=my_opt, loss="binary_crossentropy",
metrics=["acc"])
model.summary()

batch_size = 16
num_batches = int(num_samples / batch_size)
```

9. Huấn luyện mô hình.

```
for batch_num in tqdm(range(num_batches)):
    batch = X[batch_num * batch_size : (batch_num + 1) * batch_size]
    model.train_on_batch(
        batch, Y[batch_num * batch_size : (batch_num + 1) * batch_size]
    )
```

® **Bài tập (yêu cầu làm)**

5. Cho biết kết quả đánh giá mô hình qua tập test.

6. Xử lý phần mềm độc hại packer

Packer có tên VMProtect bảo vệ nội dung từ các nhà phân tích bằng các thực thi trong một môi trường ảo có kiến trúc khác lạ.

Trình packer UPX đóng gói các tập tin PE với các payload có thể bypasss anti-virus, firewall, IDS, IPS…

Vì các packer sẽ làm rối mã code sẽ làm giảm hiệu năng của các trình phân loại máy học. Bằng cách biết được trình packer nào đóng gói tập tin, ta sử dụng nó để giải mã.

® Bài tập (yêu cầu làm)

6. *Cài đặt. UPX từ https://github.com/1.upx/upx/releases, và tiến hành đóng gói các tập tin pe tại Benign PE Samples UPX*

1. Liệt kê tất cả tập tin trong thư mục.

```
import os

files_path = "Benign PE Samples UPX/"
files = os.listdir(files_path)
file_paths = [files_path+x for x in files]
```

2. Chạy upx

```
from subprocess import Popen, PIPE

cmd = "upx.exe"
for path in file_paths:
    cmd2 = cmd+" \""+path+"\""
    res = Popen(cmd2, stdout=PIPE).communicate()
    print(res)
    if "error" in str(res[0]):
        print(path)
        os.remove(path)
```

7. Xây dựng bộ phân loại packer

Giải nén tất cả các tập tin lưu trữ có tên `Benign PE Samples*.7z` vào thư mục `Benign PE Samples`, giải nén `Benign PESamples UPX.7z` vào thư mục `Benign PESamples UPX` và giải nén `Benign PESamples Amber.7z` sang một thư mục `Benign PESamples Amber`.

1. Đọc tất cả tập tin cần phân tích và gán nhãn cho chúng.

```
import os
from os import listdir

directories_with_labels = [
    ("Benign PE Samples", 0),
    ("Benign PE Samples UPX", 1),
    ("Benign PE Samples Amber", 2),
]
list_of_samples = []
labels = []
for dataset_path, label in directories_with_labels:
    samples = [f for f in listdir(dataset_path)]
    for file in samples:
        file_path = os.path.join(dataset_path, file)
        list_of_samples.append(file_path)
        labels.append(label)
```

2. Phân ra train test.

```
from sklearn.model_selection import train_test_split

samples_train, samples_test, labels_train, labels_test =
train_test_split(
    list_of_samples, labels, test_size=0.3, stratify=labels,
random_state=11
)
```

3. Import thư viện cần thiết để trích xuất N-grams.

```
import collections
from nltk import ngrams
import numpy as np
```

4. Định nghĩa hàm sử dụng trích xuất N-grams.

```
def read_file(file_path):
```

```
    """Reads in the binary sequence of a binary file."""
    with open(file_path, "rb") as binary_file:
        data = binary_file.read()
    return data


def byte_sequence_to_Ngrams(byte_sequence, N):
    """Creates a list of N-grams from a byte sequence."""
    Ngrams = ngrams(byte_sequence, N)
    return list(Ngrams)


def extract_Ngram_counts(file, N):
    """Takes a binary file and outputs the N-grams counts of its binary
sequence."""
    filebyte_sequence = read_file(file)
    file_Ngrams = byte_sequence_to_Ngrams(filebyte_sequence, N)
    return collections.Counter(file_Ngrams)


def featurize_sample(sample, K1_most_frequent_Ngrams_list):
    """Takes a sample and produces a feature vector.
    The features are the counts of the K1 N-grams we've selected.
    """
    K1 = len(K1_most_frequent_Ngrams_list)
    feature_vector = K1 * [0]
    file_Ngrams = extract_Ngram_counts(sample, N)
    for i in range(K1):
        feature_vector[i]                                          =
file_Ngrams[K1_most_frequent_Ngrams_list[i]]
    return feature_vector
```

5. Chọn N-grams mong muốn.

```
N = 2
total_Ngram_count = collections.Counter([])
for file in samples_train:
    total_Ngram_count += extract_Ngram_counts(file, N)
K1 = 100
```

```
K1_most_common_Ngrams = total_Ngram_count.most_common(K1)
K1_most_common_Ngrams_list = [x[0] for x in K1_most_common_Ngrams]
```

6. Thiết lập thuộc tính để huấn luyện.

```
Ngram_features_list_train = []
y_train = []
for i in range(len(samples_train)):
    file = samples_train[i]
    NGram_features = featurize_sample(file, K1_most_common_Ngrams_list)
    Ngram_features_list_train.append(NGram_features)
    y_train.append(labels_train[i])
X_train = Ngram_features_list_train
```

7. Huấn luyện mô hình random forest trên tập train.

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(X_train, y_train)
```

8. Thiết lập thuộc tính cho tập test.

```
Ngram_features_list_test = []
y_test = []
for i in range(len(samples_test)):
    file = samples_test[i]
    NGram_features = featurize_sample(file, K1_most_common_Ngrams_list)
    Ngram_features_list_test.append(NGram_features)
    y_test.append(labels_test[i])
X_test = Ngram_features_list_test
```

9. Sử dụng bộ phân loại được đào tạo để dự đoán trên bộ test và đánh đánh giá hiệu suất bằng confusion matrix.

```
y_pred = clf.predict(X_test)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

® Bài tập (yêu cầu làm)

7. Cho biết kết quả đánh giá.

8. MalGAN – Tạo phần mềm độc hại

Generative Adversarial Networks (GANs) – Mạng tạo sinh đối kháng có thể tạo các ví dụ về phần mềm độc hại đối đầu với các mô hình được huấn luyện và cải thiện phương pháp phát hiện của chúng. Bằng cách sửa đổi nhỏ chuỗi byte để lừa trình phân loại.

Chuẩn bị các gói cài đặt sau:

```
pip3 install pandas keras tensorflow sklearn
```

Tập dữ liệu: "MalGAN_input/samplesIn.csv" (1 là lành tín, 0 là độc hại).

1. Thêm các thư viện.

```
import os
import pandas as pd
from keras.models import load_model
import MalGAN_utils
import MalGAN_gen_adv_examples
```

2. Xác định các đường dẫn dữ liệu.

```
save_path = "MalGAN_output"
model_path = "MalGAN_input/malconv.h5"
log_path = "MalGAN_output/adversarial_log.csv"
pad_percent = 0.1
threshold = 0.6
step_size = 0.01
limit = 0.
input_samples = "MalGAN_input/samplesIn.csv"
```

3. Sử dụng GPU.

```
MalGAN_utils.limit_gpu_memory(limit)
```

4. Đọc tập tin csv chứa name và label của mẫu vào dataframe.

```
df = pd.read_csv(input_samples, header=None)
fn_list = df[0].values
```

5. Tải mô hình MalConv đã được huấn luyện.

```
model = load_model(model_path)
```

6. Sử dụng Fast Gradient Step Method (FGSM) để tạo mẫu đối kháng.

```
adv_samples, log = MalGAN_gen_adv_examples.gen_adv_samples(model,
fn_list, pad_percent, step_size, threshold)
```

7. Lưu lại log và ghi mẫu mới vào thư mục.

```
log.save(log_path)
for fn, adv in zip(fn_list, adv_samples):
    _fn = fn.split('/')[-1]
    dst = os.path.join(save_path, _fn)
    print(dst)
    with open(dst, 'wb') as f:
        f.write(adv)
```

® **Bài tập (yêu cầu làm)**

8. Cho biết kết quả đánh giá mẫu mới trong việc đánh lừa bộ nhận diện.

C. YÊU CẦU & ĐÁNH GIÁ

1. Yêu cầu

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.. Đăng ký nhóm cố định từ buổi 1.
- Sinh viên báo cáo kết quả thực hiện và nộp bài bằng **1 trong 2 hình thức**:

a) Hình thức 1 - Báo cáo chi tiết:

Báo cáo cụ thể quá trình thực hành (có ảnh minh họa các bước) và giải thích các vấn đề kèm theo. Trình bày trong file PDF theo mẫu có sẵn tại website môn học.

b) Hình thức 2 - Video trình bày chi tiết:

Quay lại quá trình thực hiện Lab của sinh viên kèm thuyết minh trực tiếp mô tả và giải thích quá trình thực hành. Upload lên **Youtube** và chèn link vào đầu báo cáo theo mẫu. **Lưu ý:** *Không chia sẻ ở chế độ Public trên Youtube.*

Đặt tên file báo cáo theo định dạng như mẫu:

[Mã lớp]-LabX_MSSV1-Tên SV1_MSSV2 –Tên SV2

Ví dụ*: [NT101.I11.1]-Lab1_14520000-Viet_14520999-Nam.*

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp báo cáo trên theo thời gian đã thống nhất tại website môn học.

2. Đánh giá:

- Sinh viên hiểu và tự thực hiện được bài thực hành, đóng góp tích cực tại lớp.
- Báo cáo trình bày chi tiết, giải thích các bước thực hiện và chứng minh được do nhóm sinh viên thực hiện.
- Hoàn tất nội dung cơ bản và có thực hiện nội dung *mở rộng – cộng điểm* (với lớp **ANTN**).

Kết quả thực hành cũng được đánh giá bằng kiểm tra kết quả trực tiếp tại lớp vào cuối buổi thực hành hoặc vào buổi thực hành thứ 2.

Lưu ý: Bài sao chép, nộp trễ, "*gánh team*", … sẽ được xử lý tùy mức độ.

HẾT

Chúc các bạn hoàn thành tốt!